Architecture-Based Verification of Software-Intensive Systems

Author: Andreas Johnsen

Mälardalen University
School of Innovation, Design and Engineering
2010 March 8
Västerås Sweden
Contents

1 Introduction ................................................. 2
   1.1 Research Topic ......................................... 2

2 Field of Research ........................................ 2
   2.1 Motivation ............................................... 2
   2.2 Problem definition ...................................... 7

3 Research Method ........................................... 8

4 The Community ............................................ 9
   4.1 Seminal Papers ........................................... 9
      4.1.1 On the Unification Power of Models ................. 9
      4.1.2 Feature-based survey of model transformation approaches .......... 9
      4.1.3 Dependability and its Threats: a Taxonomy ................. 9
      4.1.4 Deriving Tests From Software Architectures ............ 10
      4.1.5 Using Software Architecture for Code Testing .......... 10
      4.1.6 Towards Software Architecture-based Regression Testing .... 10
      4.1.7 Specification-based Test Oracles for Reactive Systems .... 10
      4.1.8 Selecting and Using Data for Integration Testing .......... 11
   4.2 Leading Research Groups ................................. 11
   4.3 Conferences ............................................ 11

5 Research Plan ............................................. 11
   5.1 Hypothesis ............................................... 11
   5.2 Research Questions ..................................... 12
   5.3 Expected Results ....................................... 12
   5.4 Research Activities .................................... 13
      5.4.1 Research courses .................................... 13
      5.4.2 Planned Publications ................................ 13
      5.4.3 Co-operations ...................................... 14
      5.4.4 Conference trips ..................................... 14
      5.4.5 Summer Schools ..................................... 14
   5.5 Timeplan ................................................ 14
   5.6 Milestones .............................................. 15

List of Figures

1 Development Process [Kat99] ................................. 3
2 Waterfall Model ............................................ 3
3 Architecture-based Development Model ........................ 4
4 Example of multiple specification languages used in software-intensive systems development .......... 5
5 Relationship between Specifications and Testing ................. 6
6 fig:Project Approach ...................................... 8
7 fig:Automating Architecture-Based Testing ..................... 8
8 High Level Project Plan .................................... 14

List of Tables

1 Overarching research challenges ............................. 6
2 Finished and ongoing Ph.D. courses .......................... 13
1 Introduction

1.1 Research Topic

Development of software-intensive systems such as embedded systems for telecommunications, avionic and automotive occurs under severe quality, schedule and budget constraints. As the size and complexity of software-intensive systems increase dramatically, the problems originating from the design and specification of the system architecture becomes increasingly significant. Architecture-based development approaches promise to improve the efficiency of software-intensive system development processes by reducing costs and time, while increasing quality. This paradox is partially explained by the fact that the system architecture abstracts away unnecessary details, so that developers can concentrate both on the system as a whole, and on its individual pieces, whether it's the components, the components' interfaces, or connections among components. The use of architecture description languages (ADLs) provides an important basis for verification since it describes how the system should behave, in a high level view and in a form where automated tests can be generated. Analysis and formal verification based on architecture specifications allow detection of problems and faults early in the development process, even before the implementation phase, thereby reducing a significant amount of costs and time. Furthermore, test cases derived from the architecture specification can later be applied to the implementation to see the conformance of the implementation with respect to the specification. However, the area of automated testing from specifications still lacks of fundamental answers, in order to ensure that automated verification and validation steps within the development process allow for greater confidence in both the process, and the final product. The research focuses on extending the theoretical knowledge within the area of ADLs and automated testing from specifications, where the extensions will be embodied in form of tools and algorithms to enable easy adoption into industrial practice, in the domains of telecommunications, avionic and automotive. Throughout this report, the terms model and specification are used interchangeably.

2 Field of Research

2.1 Motivation

Software-intensive systems are systems where software interacts with sensors, actuators, devices, other systems and people [Wir06]. Examples of such systems are embedded systems for avionic, automotive and telecommunications. What these systems have in common is that they are rapidly growing in complexity and often operating in dynamic non-deterministic environments. Because of the growing complexity of these systems, development process elements such as cost, time and quality are increasingly important. Therefore, the major question is how to optimize development processes in order to reduce cost and time without losing quality or even better, increasing quality. An optimal development process would roughly speaking be, in a technical point of view, a process where system components are created and formed to an error free system. In today’s reality, this is not the case as illustrated in Figure 1 (although the figure is derived from an old investigation, the underlying message is still up-to-date). The figure illustrates a development process of a system where introduced errors (in %), detected errors (in %) and cost of correction per error (in Deutsche Mark) are represented by three graphs. It should be mentioned that the investigation team uses the term "error" wrongly in this figure, since an error is when a system deviates from providing correct service and where the cause of an error is a fault [ALRL04]. Therefore, the correct term to use in this figure is "fault" instead of "error". As shown by the graphs, the majority of errors are introduced early in the process (note that errors are introduced before the programming phase) whereas the majority of errors are detected late in the process. Since time and consequently cost of correcting errors increase dramatically the later they are detected, prediction of possible errors is one of the main issues for developers.
The solution to the problem of costly and time consuming error correction is to somehow move the detected errors graph as close as possible to the introduced errors graph since cost of correction is low where the majority of errors are introduced, as can be seen in Figure 1. Errors and problems are detected late in development processes where developers use strategies similar to the traditional waterfall model (shown in Figure 2) where the majority of verification and validation activities take place at the end of the process [Pet09].

Architecture-based development approaches (shown in Figure 3), where a system architecture is modeled, analyzed and formally verified before implemented in order to predict if the system
will satisfy the specifications and requirements, promise to improve the efficiency of software-intensive system development processes by reducing cost and time, while increasing quality since analysis and testing of architecture models (specifications) allow detection of errors early in the process. In order to preserve the usefulness of a well modeled and verified system architecture, an implementation of the system must be implemented in conformance with the model. Consequently, architecture-based development approaches do not only deal with verification of a system model, but also testing of an implementation, to verify its conformance with the model. Architecture-based testing of the implementation is feasible by generating test cases from the architecture model which can be mapped and applied to the implementation. Test cases must be mapped to the implementation since there is a traceability problem between a model and its implementation. A system architecture abstracts away unnecessary details, so that developers can concentrate both on the system as a whole, and on its individual pieces, whether it's the components, the components' interfaces, or connections among components. System architectures can be specified (modeled) by Architecture Description Languages (ADLs) which serve as a mutual communication blueprint and provide an important base for verification and early design decisions since they describe how the systems should behave at a high abstraction level.

Figure 3: Architecture-based Development Model

Development approaches of software-intensive systems are becoming more and more closer to model-based development, since it uses tools and concepts closer to the problem area rather than those offered by "lower-abstraction-level" programming languages [SK03]. The "lower-abstraction-level" term is used here since some authors in Model-Driven Engineering (MDE) refer program code as models, because of the fact that program code is an abstraction of the underlying machine code, whereas in software engineering is the term model often referred to abstractions over program code [CH06].

Software-intensive systems are characterized by the components embedded within the systems, where these components are heterogeneous (reactive or transformative) and often have to meet real-time constraints. Because of the embedded components' heterogeneity are these software-intensive systems often multiple specified using different modeling languages, in order to capture the expected behavior of a system, at multiple levels of abstractions. These modeling languages
aim at capturing specific aspects of the system, such as functional properties, non-functional properties, component and component interactions properties, real-time properties (non-functional properties), etc., at a specific level of abstraction. However, this heterogeneous set of modeling languages lacks of coherency which is an essential obstacle to tackle [EZR+$^+$99], in order to increase the productivity of the MDE methodology. The lack of coherency among modeling languages is illustrated in Figure 4, where a label inside a rectangle denotes a concrete specification artifact and where a label inside a parenthesis denote the modeling language used to generate the specification artifact. As can be seen in the figure, model refinement across levels, from high to low level of abstraction, always include "one-to-many" refinement relations (i.e. the semantics of one element of a language must be expressed with a combination of elements in another language, in order to preserve the semantics), which means that the understanding among models is not clearly or unambiguously promoted.

Figure 4: Example of multiple specification languages used in software-intensive systems development

Because of different modeling languages and tools which captures different views of systems in different abstraction levels, there is a need for so called "model transformation" across different levels of abstraction without loss of semantics and with guaranteed traceability and consistency (in the context of specifications of embedded systems, consistency is defined as the specification being free of contradictory behavior [Oui08]), in order to support MDE, which involves architecture-based verification and validation.

Verification and validation (V&V) activities takes place at several stages of system development processes in order to increase dependability, which among others brings in concerns of confidentiality [ALRL04], in the process of development as well as the final product. V&V are most commonly carried out through the activity of formal verification (described below) and testing where testing can be classified into "white-box testing" and "black-box testing" strategies. The former approach uses an internal perspective of a system's structure and behavior to derive test cases, i.e. test cases derived based on a system's implementations; logic (source code), whereas the later approach uses an external perspective of a system's structure and behavior to derive test cases, i.e. test cases derived based on a system's specification artifacts [MSB04]. Black-box testing is a central strategy (yet, it is a strategy that must be progressed in the MDE methodology) of importance since development of software-intensive systems progresses to higher level of system designs, as a consequence of the increasing complexity of systems, where white-box testing
becomes an increasingly complicated (and thus increasingly difficult since white-box testing is a human-intensive/knowledge-intensive activity) and inefficient strategy. Furthermore, black-box testing provides an opportunity to alleviate the limitations of white-box testing, since the former strategy uses specification artifacts, generated at several steps of the development process, as basis for carrying out testing activities as shown in Figure 5.

![Figure 5: Relationship between Specifications and Testing](image)

Formal verification is about verifying the correctness of a system model with respect to a formal specification by using formal methods of mathematics (i.e. "formal reasoning about the system"), and is of great importance in order to improve system development processes, since it provides an opportunity to predict problems and faults in the system [Bje05]. The most common approach to formal verification is called "model checking" which is exhaustive exploration of a mathematical model.

Despite that a system has been developed with efficiently effective V&V, as much as one-third of the total cost of a system’s lifetime can be accounted for regression testing (retesting a modified system), where architecture specifications has the potential to reduce this cost significantly [MDR05].

Nevertheless, formal verification, black-box testing and regression testing must be automated in order to be efficient and effective for industrial use, which is an difficult obstacle to tackle since automating black-box testing is dependent on four factors: traceability between specifications; the quality of the specifications used to generate test cases; existence of a test harness (automatically generated from specifications) that generates and coordinates the testing process; and the ability to use specifications as oracles to determine the accuracy of the testing activities.

This section can be synthesized into two overarching challenges:

<table>
<thead>
<tr>
<th>Table 1: Overarching research challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Challenge 1</strong></td>
</tr>
<tr>
<td><strong>Challenge 2</strong></td>
</tr>
</tbody>
</table>
2.2 Problem definition

This section presents the tasks that need to be finished in order to master the overarching challenges addressed in previous section, where the tasks are presented respectively to the challenges.

**Task 1**: Support model transformation without loss of meaning and with guaranteed traceability and consistency

The Architecture Analysis and Description Language (AADL), by Feiler et al. [FGH06] [HF07], was released and published as a Society of Automotive Engineers (SAE) Standard AS5506, in November 2004. AADL, which is derived from the ADL MetaH [Dis05], is a textual and graphical language used to model, specify and analyze software- and hardware-architectures of real-time, embedded and high dependability systems. The AADL language is based on a component-connector paradigm that describes components, component interfaces and the interaction between components. A system is modeled as a hierarchy of components where components that represent the application software are mapped onto the components that represent the hardware platform. Properties can be associated with different AADL elements (e.g. components, subcomponents, features, connections etc.) to provide descriptive information about the respective AADL element. Changes to the runtime architecture can be described by modes and transitions of modes. Timing and performance are modeled by explicit properties (such as deadlines, worst-case execution time, arrival rate, period etc.) associated with components and defined concurrency and interaction semantics. Detailed description about abstract paths of information through a system can be defined and analyzed as a flow path.

The AADL's features described above coupled with the ability to meet most of the qualities of an ideal ADL [MT00] (comparison of ADLs where conclusions are based on MetaH) make it the architectural specification language of choice for this research. Currently, the AADL semantics are thoroughly and well understood but not formally specified [Dis05]. This issue is critical in the context of model transformation, where it must be possible to formally analyze the transformation process for completeness and consistency, in order to ensure that there are no loss of semantics. We solve this issue by formally define semantics for a subset of the AADL language using the Timed Abstract State Machine language (TASM) [Oui06], which is a language built on the automata theory extended with resource and timing aspects. The AADL subset is chosen in consideration to the AADL constructs that describe critical functional and non-functional aspects of safety-critical systems. By formally define the semantics for the SPARKAda language, in similarly way as with the AADL language, we can create a common formal underpinning among the various specifications in order to ensure minimal loss of semantics, and that any loss of semantic is explicitly described.

**Task 2**: Automate Architecture-based V&V

The system architecture specification provides an important basis for the integration testing phase, as shown in Figure 5. Eckelmann and Richardson [ER96] describes two primary goals of integration testing; ensuring consistency of component interfaces; and ensuring that data- and control-interactions (data-flow and control-flow respectively) between components are correct. In order to perform automated formal verification and test case generations from AADL specifications, it is critical to have the ability to execute the specifications, where an transformation to the TASM language solve this problem since it is executable. A tool will be created for completeness and consistency checking (specification having response for every possible class of input and specification being free of contradictory behavior, respectively [Oui08]) of AADL specifications and by defining formal semantics of a subset of AADL, we will through executables perform dynamic model checking of the specification, in order to verify the architecture specification for correctly data- and control-flows. Additionally, the tool will have features that creates test suites that can be used for regression testing (retesting evolved/modified systems).
3 Research Method

In the previous sections, we described the challenges associated with software-intensive system development and highlighted two critical areas: model transformation without loss of semantics; and automated architecture specification-based V&V. We believe that these two areas must be addressed in parallel in order to overcome the challenges, as shown in Figure 6.

Both AADL and TASM tools have the possibility to create XML documentation, which provides means of verifying the accuracy of the semantics during model transformation to ensure completeness and consistency. The effectiveness of model transformation efforts can further be analyzed by defining semantics for a subset of the SPARK language.

Automating architecture-based V&V has four key challenges: data generation, data selection, test suite generation and oracle creation where these elements are controlled and coordinated by a test harness. The proposed method to solving the problem of automating architecture-based V&V is shown in Figure 7. The sources of test data are derived from both the TASM language's data specification construct and the AADL data specification construct. The data selection problem is solved by extending the idea of category partitioning [OB88] to the architecture-level. Static and dynamic analysis of the specification will be used as a means of generating integration test sequences. A TASM specification will then be executed (simulated) based on integration test sequences, where the outcome of the execution will be used to determine pass-fail criteria. The integration test case generation algorithms will be compared to a control-flow and data-flow graph, generated by an translation from the TASM specification, in order to measure the effectiveness of the algorithms using traditional coverage metrics.

The smallest subset of a regression test suite that encapsulates the same coverage of a modified/evolved system, as the complete regression test suite, will be determined by extending Richardson's concept of specification slicing [CR94]. The subset is determined by comparing both the original system’s and the modified system’s graphs, which captures the systems’ control- and data-flows, to identify new or modified nodes, in order to determine which test cases are unnecessarily to retest on the modified system (since the modification does not affect those nodes of the system that has not been changed).
This section can be summarized into high level tasks that have to be done in order to achieve the objectives of the research:

- Detailed literature review on software testing, focused on model based testing and integration testing
- Explore the Model Driven Engineering (MDE) area in order to get familiar with concepts, methodologies and state-of-the-art research related to model transformation and model based testing
- Derive knowledge about appropriate testing frameworks and develop testing skills
- Study the Architecture Analysis and Design Language (AADL)
- Study different tools/languages such as TASM, Spark, Uppaal and Times for modeling and verification purposes
- Develop algorithms for architecture-based verification and regression testing

4 The Community

This section presents relevant papers, research groups and conferences within the research area.

4.1 Seminal Papers

4.1.1 On the Unification Power of Models

J. Bezivin proposes a vision of MDE development in paper [Bez05], based on experience in the development of object technology. From the main message driving the object technology "everything is an object", the new message driving the MDE technology "everything is a model" is analyzed for interesting properties. The paper concentrates on postulating that the main message "everything is a model" is associated with two core relation, representation and conformance. The proposed vision of MDE is useful in order to understand the MDE area in general and the Model Driven Architecture (MDA), which is an model-driven approach defined by the Object Management Group (OMG).

4.1.2 Feature-based survey of model transformation approaches

K. Czarnecki and S. Helsen propose in [CH06] a framework for the classification of existing and proposed model transformation approaches. The framework makes explicitly the different design choices for model transformations and based on their analysis of the different choices, they propose major categories in which most approaches are included. The paper is helpful in the way that it helps developers making the right choice of model transformation approach.

4.1.3 Dependability and its Threats: a Taxonomy

B. Randell et al. provides in paper [ALR04] the main definitions relating to dependability which is a generic concept including attributes such as reliability, availability, safety, confidentiality, integrity and maintainability. Furthermore, the threats to dependability (faults, errors and failures) as well as the attributes of dependability are addressed through the definitions. The paper is useful since it describes the importance of dependable safety-critical systems and how the dependability is achieved.
4.1.4 Deriving Tests From Software Architectures

Jin Z. and Offit J. present in [JO01] an architecture-based testing technique to test software. Software architectures abstract away details from applications so that the applications can be viewed as sets of components with connectors that describe the interactions among components. Architecture description languages are used to model software architecture for analysis and development. The authors distinguish architecture testing from system testing because system testing tests the overall system to see if it meets its requirements while architecture testing attempts to test the interactions/relations among components at the architecture level. Their approach defines general testing criteria that define the test requirements, which are used to generate test inputs. Their technique generates test criteria from traditional data-flow and control-flow criteria by using properties from data flow and control flow, at the architecture-level. The properties are data flow reachability, control flow reachability, connectivity and concurrency. This paper is useful since it describes how architectures can be integration tested when their architecture-based criteria are applied to an architecture-specification.

4.1.5 Using Software Architecture for Code Testing

In paper [MBI04], Miccini H. et al. presents a fully developed approach, which uses a software architecture as a reference model for testing the conformance of the implementation with respect to the architecture specification (architecture reference model). Their goal is "to provide a test manager with a systematic method to extract suitable test classes for higher levels of testing and to refine them into concrete tests at the code level". Their approach to software architecture conformance testing is based on graphs of labeled transition systems that captures architectural behaviors, which is used for generating tests that can be converted to code-level conformance tests. This paper is useful since it describes how the conformance of the implementation with respect to the specification can be tested by generating tests from automata systems.

4.1.6 Towards Software Architecture-based Regression Testing

In paper [MDR05], Muccini H. et al. explores how regression testing, at the software-architecture level, can be systematically used in order to reduce cost of retesting modified systems. In general, the authors describe two goals, 1) how existing implementation-level test cases can be reused to test the conformance between modified code and the architectural specifications, and 2) how to reuse architecture-level test cases to test the conformance of the source code with respect to the evolved software architecture. Their approach to address both goals relies on integrating code-level regression testing with architecture-based regression testing. They use selective testing technique for code-level regression testing and use the same logical steps for architecture-based regression testing. The method used for regression testing is based on comparing nodes of graphs, where one graph represents a program and another represents a modified version of the original program. This paper is useful since it provides an approach for regression testing at the architecture-level by comparing graphs, which is one goal of the tool we want to develop.

4.1.7 Specification-based Test Oracles for Reactive Systems

Output from a test execution is typically examined by someone visually to determine if the system behaved correctly. In paper [RAO92], Richardson, D. J. et al. presents a specification-based testing approach that they are developing, to be applicable on several language paradigms and several specification languages. The approach is to derive test oracles from multi-specifications in conjunction with testing to represent an oracle for each test class specified by a testing criterion. This paper is useful since it provides an approach of how oracles can be derived from specifications, which is something we want to do with TASM specifications.
4.1.8 Selecting and Using Data for Integration Testing

Harrold M.J and Sofia M.L describe in [HS91] a testing tool for integration testing based on data-flow testing. Integration testing is testing of a modular program that contains procedures that interact with other procedures. Not only does the program’s procedures need to be unit tested, errors can still exist in the procedures interfaces. Their approach for integration testing is to extend dataflow testing (definition-use pairs) to inter-procedural testing by developing both an analyze technique and a testing technique. This paper is important since it describes an approach of how to select data for integration testing.

4.2 Leading Research Groups

Prof. Jean Bezivin, Faculty of Sciences, University of Nantes, France  The group focuses its research on several aspects of model-driven engineering and model transformation

Prof. Paul Pettersson, Real-Time Modeling and Analysis, Malardalen University, Sweden  The group focuses its research on formal modeling, analysis and verification of real-time embedded systems

4.3 Conferences

- ICSE (International Conference on Software Engineering): Conference which "is the premier software engineering conference, providing a forum for researchers, practitioners and educators to present and discuss the most recent innovations, trends, experiences and concerns in the field of software engineering" [http://www.icse-conferences.org/]

- ICST (International Conference on Software Testing, Verification and Validation): a leading testing conference "has been very successful in bringing industry and research together to help shape the future of testing. ICST conferences have raised and provoked many new important issues and challenges for this area, driving and focusing research activities, as well as, the evolution of testing practices within industry" [http://vps.it-sudparis.eu/icst2010/]

- ASE (International Conference on Automated Software Engineering): The conference concerns analysis, design, implementation, testing, and maintenance of large software systems. "Automated software engineering focuses on how to automate or partially automate these tasks to achieve significant improvements in quality and productivity. ASE 2010 encourages the submission of technical research papers, experience reports, demonstrations, and short papers about emerging topics. In addition it features a doctoral symposium, several associated workshops, and a number of in-depth tutorials" [http://soft.vub.ac.be/ase2010/]

- ICMT (International Conference on Model Transformation): conference focusing in challenges for model transformation technology

- ISSTA (International Symposium on Software Testing and Analysis): "is the leading research conference in software testing and analysis, bringing together academics, industrial researchers, and practitioners to exchange new ideas, problems, and experience on how to analyze and test software systems. The ISSTA program will include technical papers, keynotes, workshops, and a doctoral symposium." [http://selab.fbk.eu/issta2010/]

5 Research Plan

5.1 Hypothesis

Following hypothesis represent my perspective of what the research project should consider when it comes to architecture-based verification of software-intensive systems:
Architecture-based verification of software-intensive systems can improve the efficiency and effectiveness of the complete development life-cycle of software-intensive and high-integrity systems, through a single holistic framework.

5.2 Research Questions

The main question to address (which is related to the hypothesis described above) during my research is: How to improve the efficiency and effectiveness of the complete development life-cycle of software-intensive and high-integrity systems, through a single holistic framework? A decomposition of this question, in order to emphasize its attributes:

Q1: How to formally specify AADL semantics, by using the TASM language?

Q2: How to perform model transformations, from an AADL specification to a TASM specification, without loss of semantics and with guaranteed traceability and completeness?

Q3: How to automate architecture-based verification and validation?

Q3.1: How to perform automated formal verification of TASM specifications (indirectly of AADL specifications), through the activity of model checking, based on integration testing methodologies?

Q3.2: How to perform automated architecture-based conformance testing of the implementation with respect to its architecture specifications?

Q3.2: How to perform efficient, effective and automated architecture-based regression testing?

5.3 Expected Results

The project involves creation of an integrated framework for specifying and verifying software-intensive and high-dependable systems, based on integrating research in specification languages and research in software testing. This framework will make two theoretical contributions:

1. Formally specified semantics of a subset of both the AADL and SPARK languages, by using the TASM language

2. Automated architecture-based verification and validation algorithms addressing integration and regression testing

These contributions will be the result of extensions of the TASM tool-set, where the tool-set will be extended with capabilities to accept AADL specifications and with embodied and automated verification and validation algorithms. Research problems regards problems within three industries: aerospace, automotive and telecommunications and the results from the research will enhance innovation and improve the efficiency and effectiveness of the system development processes, within these industries.
5.4 Research Activities

5.4.1 Research courses

Finished and ongoing courses are shown in the following table:

<table>
<thead>
<tr>
<th>Course Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling and Verification of Real-time System</td>
<td>Finished</td>
</tr>
<tr>
<td>Safety Critical Systems' Engineering</td>
<td>Ongoing</td>
</tr>
<tr>
<td>Advanced Model-Driven Engineering course</td>
<td>Ongoing</td>
</tr>
<tr>
<td>Research Planning</td>
<td>Ongoing</td>
</tr>
</tbody>
</table>

5.4.2 Planned Publications

This section depicts planned publications where the depicted publications are chronologically ordered, starting from nearest future.

1. **Paper A** - In my master thesis, an airplane control system was specified using the Architecture Analysis and Description Language (AADL). This specification served as a starting point of a system development process where derived architecture-based verification algorithms were applied. The thesis was done in order to extend the knowledge base within the area of verification from specifications and a paper summarizing the work would give important feedback.

2. **Paper B** - This paper will be a result of the first milestone, where a subset of the AADL semantics has been formally specified. The chosen subset will be outlined as well as an explanation of why the specific subset has been chosen. Furthermore, the technique used to formally specify the language will be outlined.

3. **Paper C** - Developed architecture-based verification algorithms based on integration testing will be the topic of this paper as well as case studies evaluating their effectiveness and efficiency. The algorithms will include formal verification of systems' architecture specifications and, in order to test the conformance of an implementation with respect to its specifications, test case generation and mapping to the implementation. The paper will also include ideas of how the algorithms can be integrated automated by extensions of the TASM tool-set.

4. **Paper D** - This paper will be done similarly as paper "A", except that the SPARK languages will be the discussed language in this paper.

5. **Paper E** - Developed architecture-based regression testing algorithms based on data flow and control flow graphs will be the topic of this paper. Similarly as paper B, the paper will include ideas of how the algorithms can be integrated automated by extensions of the TASM tool-set.

6. **Paper F** - This paper will be the result of the integration of algorithms and features to the TASM tool-set, i.e. developed TASM tool-set extension to accept AADL specifications and features for automated architecture-based verification and regression testing. The paper will present an overview of the complete TASM tool-set and case studies evaluating its efficiency and effectiveness.
5.4.3 Co-operations

The current and most probably future co-operations are mainly provided by my membership in the SWELL research school. SWELL is a Swedish research school in V&V and brings together Ph.D. members (currently 11 Ph.D.), industries and universities. SWELL's goals are to:

- Educate international top class Ph.D.s in software testing, verification and validation
- Ensure closer connections and interchange of knowledge and ideas between industry, companies and Swedish leading V&V academics
- Drive the national innovation in V&V

5.4.4 Conference trips

Connect papers to appropriate conferences mentioned in THIS report

5.4.5 Summer Schools

At 21-25 June, 2010, I will participate to TAROT (Training And Research On Testing) summer school in Austria. TAROT is "a network created to foster the mobility of students, faculty members and research scientists working in the field of testing of software and communication systems. This summer school brings together lecturers, researchers, students and people from the industry across Europe for one week of presentations, discussions and getting to know each other. The main goal of the TAROT Summer School is to give researchers and particularly Ph.D. students the opportunity to follow a number of tutorials or invited talks by key experts in the field."

5.5 Timeplan

An high level project plan is shown in Figure 8. The first year of research will be dedicated to, except for background research of the research area, formal definition of a subset of AADL semantics using the TASM language. The year will also be dedicated to development of integration testing algorithms which will be evaluated in case studies. The second year will semantics for a subset of the SPARK language be formally specified, using the TASM language, and algorithms for regression testing will be developed and evaluated by case studies. The third and last year of the time plan will be dedicated to extend the TASM tool set with the integration testing and the regression testing algorithms as well as the ability to accept specification written in AADL.

![Figure 8: High Level Project Plan](image)
5.6 Milestones
The major milestones in my research are submissions of the papers presented in Section 5.4.2:

Summer 2010, Submission of paper A

Autumn/Winter 2010, Submission of paper B

Spring/Summer 2011, Submission of paper C

Autumn/Winter 2011, Submission of paper D

Spring/Summer 2012, Submission of paper E

Autumn/Winter 2012, Submission of paper F
References


